91.304 Foundations of (Theoretical) Computer Science

Chapter 1 Lecture Notes (Section 1.4: Nonregular Languages)

David Martin
dm@cs.uml.edu

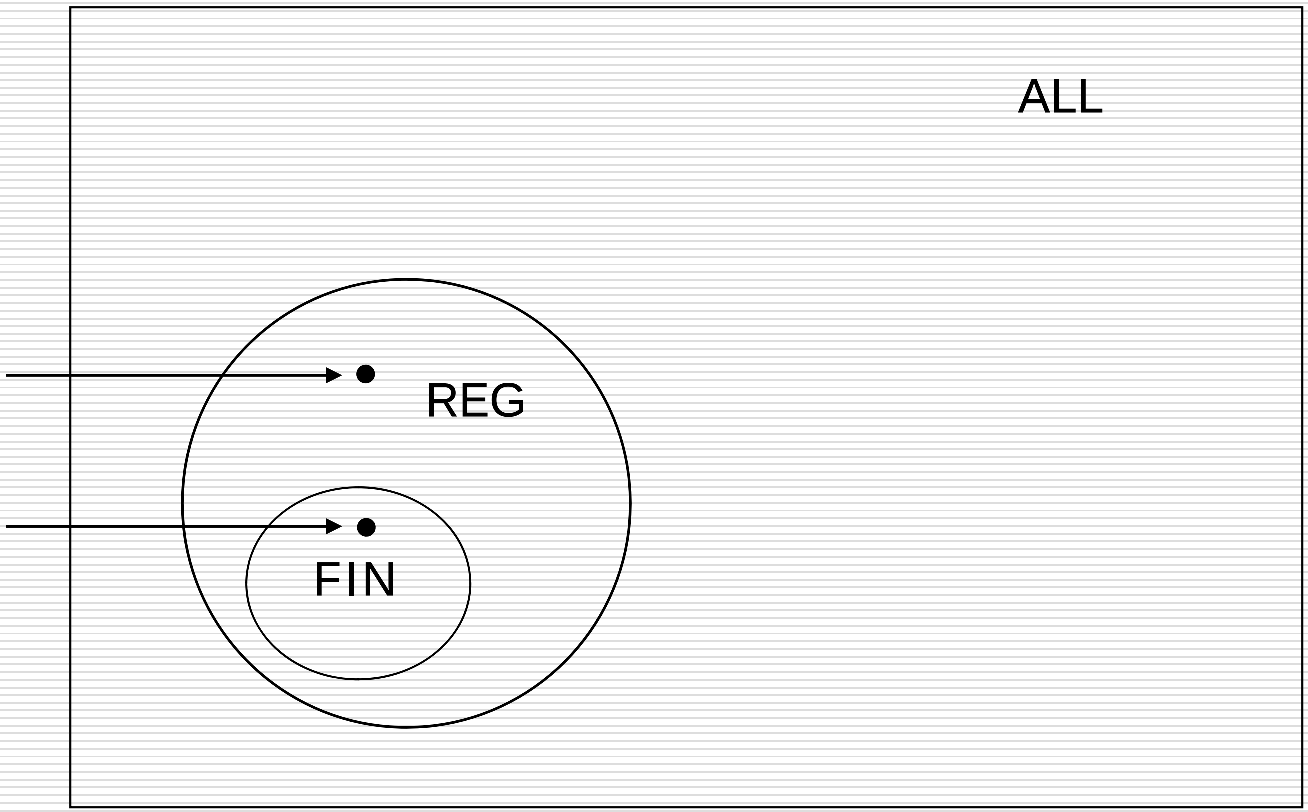
With some modifications by Prof. Karen Daniels, Fall 2014

Picture so far

Each point is a language in this Venn diagram

0*(101)*

{ 0101, ε }

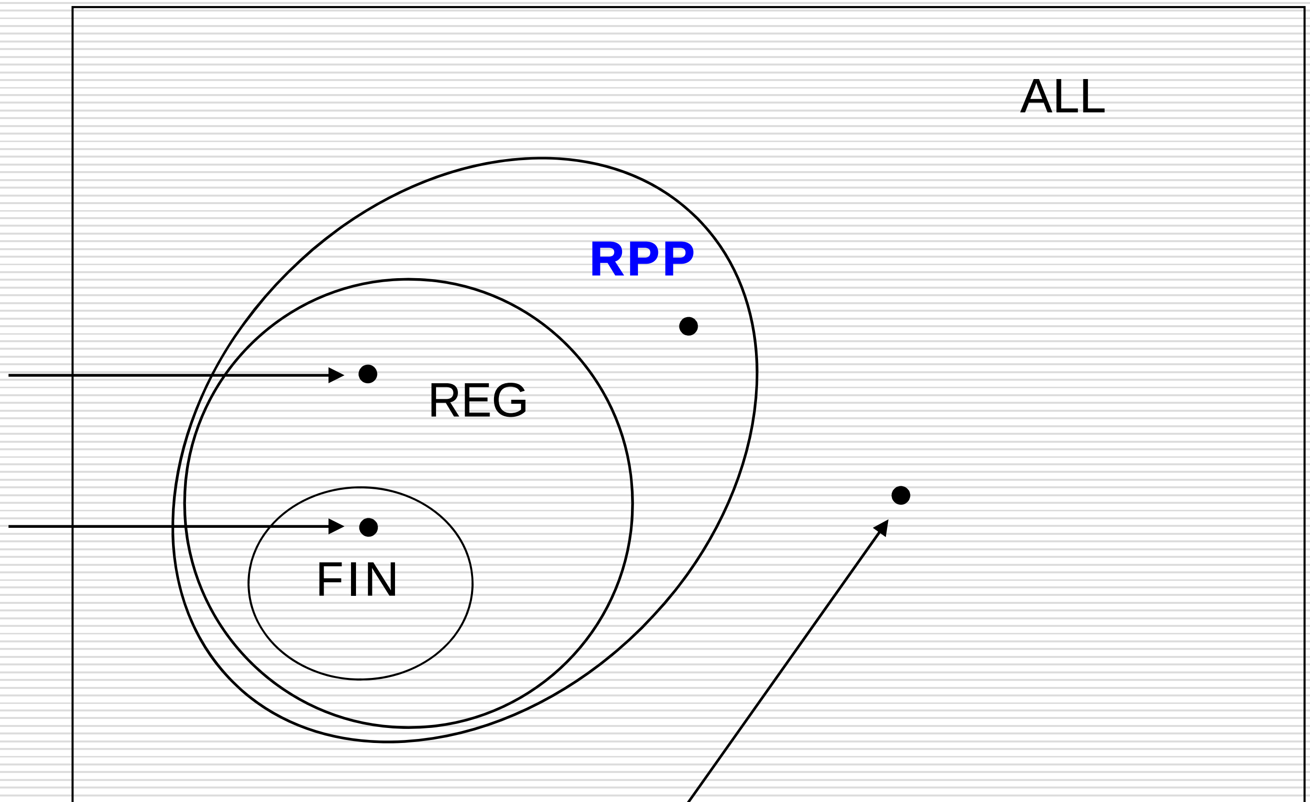


Where we are heading now...

Each point is a language in this Venn diagram

$0^*(101)^*$

$\{ 0101, \varepsilon \}$



$B = \{ 0^n 1^n \mid n \geq 0 \}$

§1.4 Nonregular languages

- For each possible language L of strings over Σ,
  - $\emptyset \subseteq L$. So $\emptyset$ is the <u>smallest</u> language. And $\emptyset$ is regular
  - $L \subseteq \Sigma^*$. So Σ^* is the "<u>largest</u>" language of strings over Σ. And Σ^* is regular.
- Yet there are languages <u>*in between*</u> these two extremes that are <u>not</u> regular

A nonregular language

$B = \{ 0^n 1^n \mid n \geq 0 \}$

$= \{ \varepsilon, 01, 0011, 000111, \cdots \}$

is not regular.

- Why?
  - Q: how many bits of memory would a DFA need in order to recognize B?
  - A: there *appears* to be no single number of bits that's big enough to work for every element of B.
    - Remember, the DFA needs to reject all strings that are not in B.

Other examples

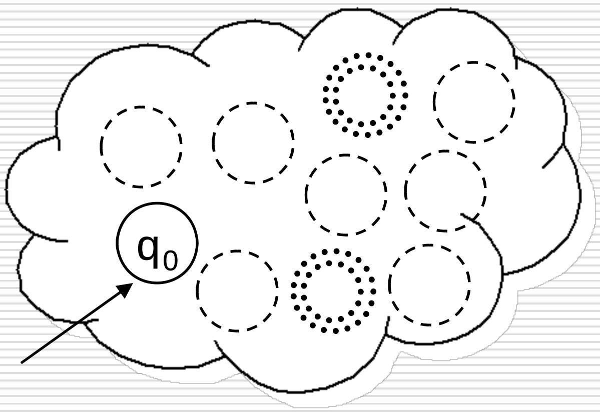
- $C = \{ w \in \{0,1\}^* \mid n_0(w) = n_1(w) \}$
  - Needs to count a potentially unbounded number of '0's… so nonregular
- $D = \{ w \in \{0,1\}^* \mid n_{01}(w) = n_{10}(w) \}$
  - Needs to count a potentially unbounded number of '01' substrings… so ??
- Need a technique for establishing nonregularity that is more formal and… less intuitive?

Proving nonregularity

- To *prove* that a language is nonregular, you have to show that *no DFA whatsoever* recognizes the language
  - Not just the DFA that is your best effort at recognizing the language
- The *pumping lemma* can be used to do that
- The pumping lemma says that every regular language satisfies the "regular pumping property" (RPP)
  - Given this, if we can show that a language like B doesn't satisfy the RPP, then it's not regular
  - $B = \{ 0^n 1^n \mid n \geq 0 \}$

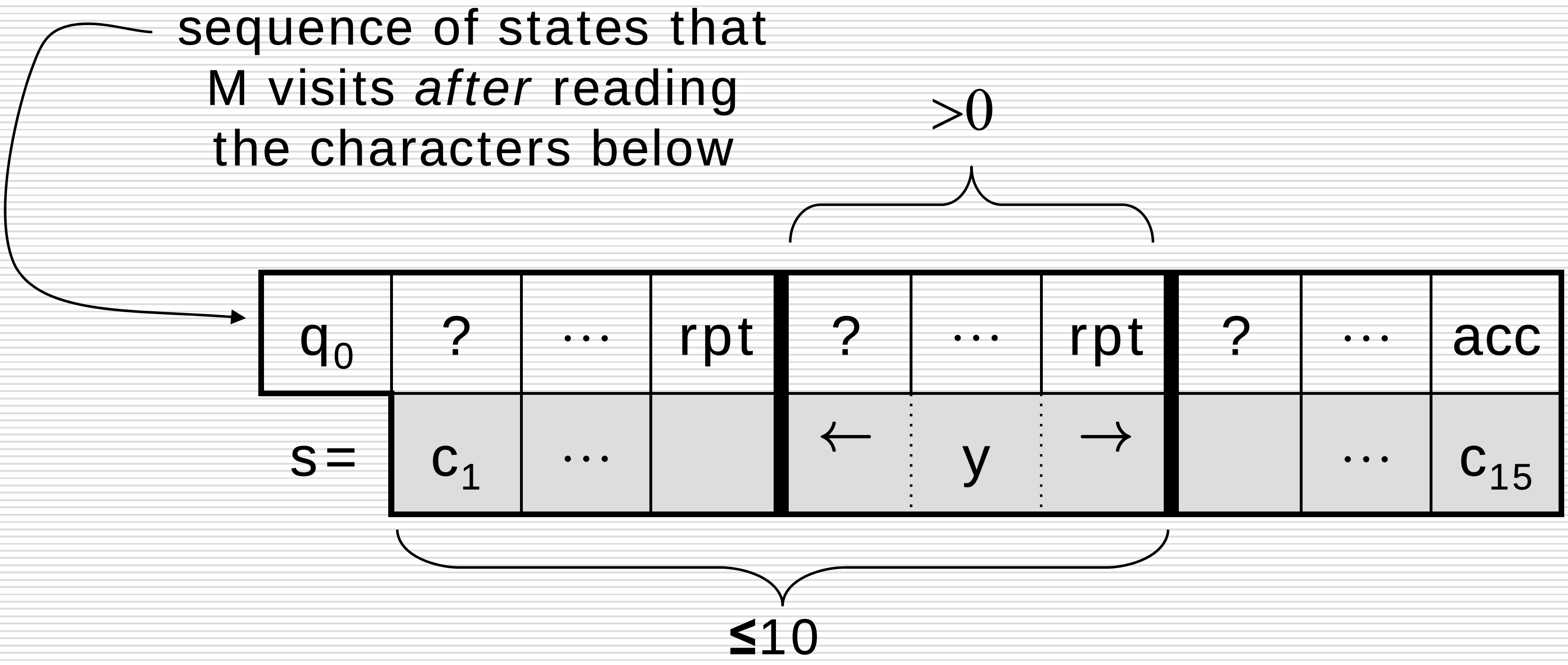
Pumping lemma, informally

- Roughly: "if a regular language contains any 'long' strings, then it contains infinitely many strings"
- Start with a regular language and suppose that some DFA $M=(Q,\Sigma,\delta,q_0,F)$ for it has $|Q|=10$ states.
- What if M accepts some particular string s where $s=c_1c_2\cdots c_{15}$ so that $|s|=15$?



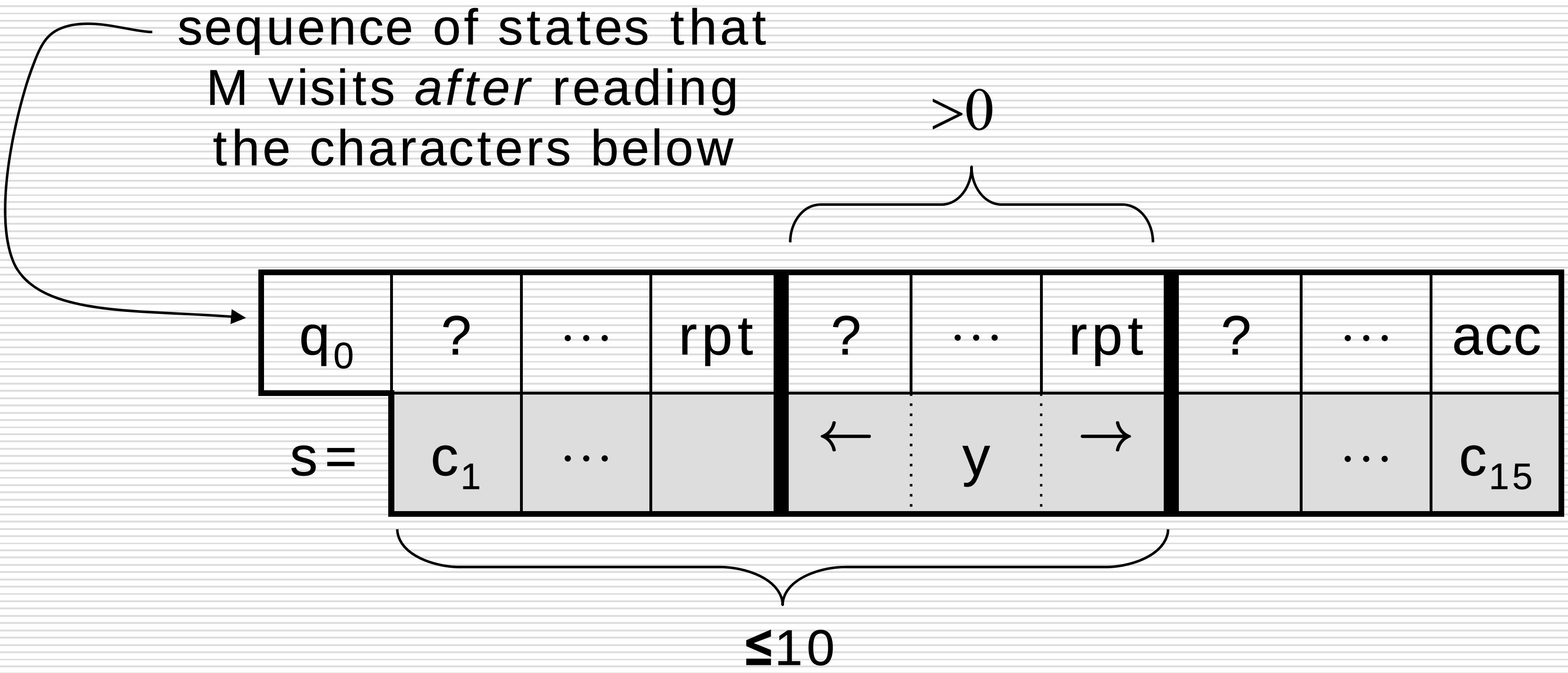
Pigeonhole principle

- With 15 input characters, the machine will visit *at most* 16 states
  - But there are only 10 states in this machine
  - So clearly it will visit at least one of its states more than once
    - Let **rpt** be our name for the first state that is visited multiple times on that particular input s
    - Let **acc** be our name for the accepting state that s leads to, namely, $\delta^*(q_0,s)$ = acc
      - **δ^*(q,x) is the set of all states reachable in the machine after starting in state q and reading the entire string x**
    - Let **y** be our name for the leftmost substring of s for which δ^*(rpt, y)=rpt
      - Since there are no ε transitions in a DFA, a state being "visited multiple times" means that it read at least one character. **Therefore, |y| > 0**

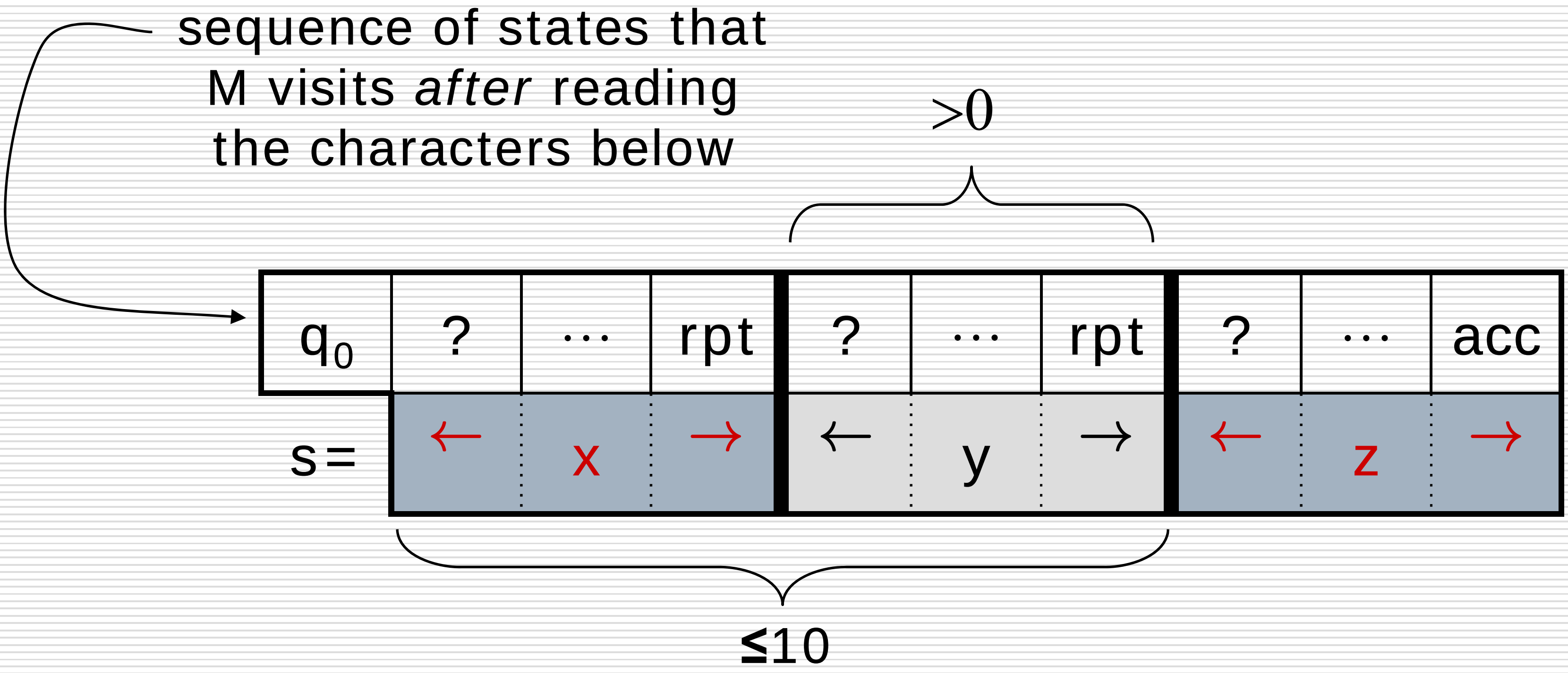


After reading $c_1 \cdots c_{10}$ (first 10 chars of s), M must have already been to state rpt and returned to it at least once… because there are only 10 states in M.

Of course the repetition could have been encountered earlier than 10 characters too…



Assigning new names to the pieces of s...



Assigning new names to the pieces of s...

So s = xyz as shown above.

With these names, the other constraints can be written

$|y| > 0$

$|xy| \leq 10$

M accepts other strings too

q_0	?	⋯	rpt	?	⋯	rpt	?	⋯	acc
s=	←	x	→	←	y	→	←	z	→

- Consider the string xz

M accepts other strings too

q_0	?	$\cdots$	rpt	?	$\cdots$	acc
s=	$\leftarrow$	x	$\rightarrow$	$\leftarrow$	z	$\rightarrow$

- Consider the string xz
  - $\delta^*(q_0,x)$ = rpt
  - δ^*(rpt,z) = acc (from previous slide)
  - So xz $\in$ L(M) too

M accepts other strings too

q_0	?	$\cdots$	rpt	rpt	rpt	?	$\cdots$	acc
s=	$\leftarrow$	x	$\rightarrow$	y	y	$\leftarrow$	z	$\rightarrow$

- Consider the string xyyz
  - $\delta^*(q_0,xy)=rpt$ (from 2 slides ago)
  - $\delta^*(rpt,y)=rpt$ (from same previous result)
  - $\delta^*(rpt,z)=acc$ (from same previous result)
  - So $xyyz \in L(M)$ also
- Apparently we can repeat y as many times as we want

p-regular-pumpable **strings**

- **Definition** (not in textbook) A **string** s is said to be *p-regular-pumpable in a language* L $\subseteq \Sigma^*$ if there exist x,y,z $\in \Sigma^*$ such that
  1. s=xyz ("x,y,z are a decomposition of s")
  2. $|y|>0$
  3. $|xy| \leq p$
  4. For all $i \geq 0$,
     $x\, y^i\, z \in L$ ("the y part of s can be pumped to produce other strings in the language")
  - It follows that s *must* be a member of L for it to be p-pumpable
- The 15-character string s in the previous example was 10-regular-pumpable in L(M).
  - Is s also 15-regular-pumpable?

p-regular-pumpable **languages**

- **Definition** A language L *is p-regular-pumpable if*
  - for every $s \in L$ such that $|s| \geq p$,

    the string s is p-pumpable in L
  - in other words, "every long enough string in L is pumpable"
- Our previous example **language** was 15-regular-pumpable
  - Is it also 10-regular-pumpable?

RPP(p) and RPP

- **Definition** RPP(p) is the class of languages that are p-regular-pumpable. In other words, RPP(p) = { $L \subseteq \Sigma^*$ | L is p-regular-pumpable}
- **Definition** RPP is the class of languages that are p-regular pumpable for some p. In other words,

$$\mathrm{RPP} = \bigcup_{p=0}^{\infty} \mathrm{RPP}(p)$$

- Lots of notation and apparent complexity, but the idea is simple: RPP is the class of languages in which every **sufficiently long** string is pumpable

Pumping lemma

Theorem 1.70 (rephrased) If $L \subseteq \Sigma^*$ is recognized by a p-state DFA, then $L \in RPP(p)$

Proof Just like our example, but use p instead of the constant 15 (or number of states = 10 in our example)

Corollaries

Primary application of Pumping Lemma

- REG $\subseteq$ RPP
- If $L \notin$ RPP then $L \notin$ REG.

Proving a language nonregular

- First unravel these definitions, but it amounts to proving that L is not a member of RPP. Then it follows that L isn't regular
  - Proving that L isn't in RPP allows you to concentrate on the *language* rather than considering all possible *proposed programs* that might recognize it

Unraveling RPP: a direct rephrasing

Rephrasing L is a member of RPP if

- There exists $p \geq 0$ such that
  - For every $s \in L$ satisfying $|s| \geq p$,
    - There exist $x,y,z \in \Sigma^*$ such that
      1. $s=xyz$
      2. $|y|>0$
      3. $|xy| \leq p$
      4. For all $i \geq 0$,

         $x\,y^i\,z \in L$

$(\exists$ p$)$ $(\forall$ s$)$ $(\exists$ x,y,z$)$ $(\forall$ i$)$!!!

Pretty complicated

Nonregularity proof by contradiction

Claim Let B = { $0^n 1^n$ | $n \geq 0$ }. Then B is not regular

Proof We show that B is not a member of RPP by contradiction.

So assume that $B \in$ RPP (and hope to reach a contradiction soon). Then there exists $p \geq 0$ associated with the definition in RPP.

We let $s = 0^p 1^p$. (Not the exact same variable as in the RPP property, but an example of one such possible setting of it.) Now we know that $s \in B$ because it has the right form.

Proof continued

- Now $|s| = 2p \geq p$. By assumption that $B \in$ RPP, there exist x,y,z such that
  1. $s=xyz$ ($= 0^p 1^p$, remember)
  2. $|y|>0$
  3. $|xy| \leq p$
  4. For all $i \geq 0$,

     $x\ y^i\ z \in B$
- Part (3) implies that $xy \in 0^+$ because the first p-many characters of s=xyz are all 0
  - So y consists solely of '0' characters
  - … at least one of them, according to (2)

Proof continued

- But consider:
  - $s = xyz = xy^1z = 0^p\ 1^p$ (where we started)
  - y consists of one or more '0' characters
  - so xy^2z contains more '0' characters than '1' characters. In other words,
    - $xy^2z = 0^{p+|y|}\ 1^p$
  - so $xy^2z \notin B = \{ 0^n\ 1^n \mid n \geq 0 \}$.
- This contradicts part (4)!!
- Since the contradiction followed *merely* from the *assumption* that B∈RPP (and right and meet and true reasoning about which we have no doubt), that assumption must be wrong **QED**

Observations

- We needed (and got) a contradiction that was a necessary consequence of the assumption that B $\in$ RPP and then relied on the Theorem 1.70 corollaries
- RPP mainly concerns strings that are longer than p
  - So you should concentrate on strings longer than p...
  - even though p is a variable. But clearly $|0^p1^p|>p$
- In our example we didn't "do" much: after our initial choice of s and thinking about the implications we found a contradiction right away
  - Many other choices of s would work, but *many don't*, and even some that do work require more complex arguments—for example, $s=0^{\lfloor p/2\rfloor+1}1^{\lfloor p/2\rfloor+1}$
  - Choosing s wisely is usually the most important thing

Game theory formulation

- The direct (non-contradiction) proof of non-context-freeness can be formulated as a two-player game
  - **You** are the player who wants to establish that L is not in RPP
  - Your **opponent** wants to make it difficult for you to succeed
  - Both of you have to play by the rules

Game theory continued

The game has just four steps.

1. Your **opponent** picks $p \geq 0$
2. **You** pick $s \in L$ such that $|s| \geq p$
3. Your **opponent** chooses $x,y,z \in \Sigma^*$ such that $s=xyz$, $|xy|>0$, and $|xy| \leq p$
4. **You** produce some $i \geq 0$ such that $xy^iz \notin L$

Game theory continued

- If you are able to succeed through step 4, then you have won only one round of the game
- To show that a language is not in RPP you must show that you can **always** win, regardless of your opponent's legal moves
  - Realize that the opponent is free to choose the most inconvenient or difficult p and x,y,z imaginable that are consistent with the rules

Game theory continued

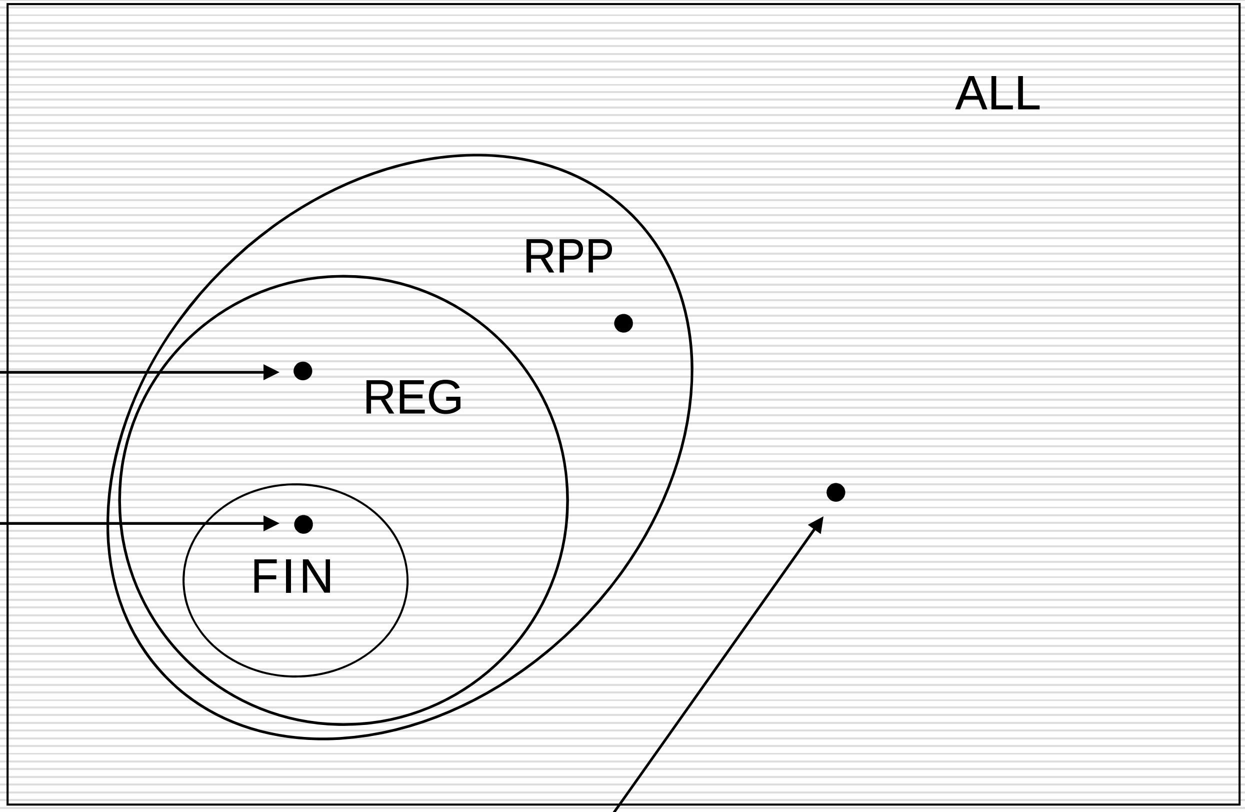
- So you have to present a *strategy* for always winning — and convincingly argue that it will always win
  - So your choices in steps 2 & 4 have to depend on the opponent's choices in steps 1 & 3
  - And you don't know what the opponent will choose
  - So your choices need to be framed in terms of the variables p, x, y, z

Picture so far

Each point is a language in this Venn diagram

$0^*(101)^*$

$\{\ 0101, \varepsilon\ \}$



$B = \{\ 0^n 1^n \mid n \geq 0\ \}$